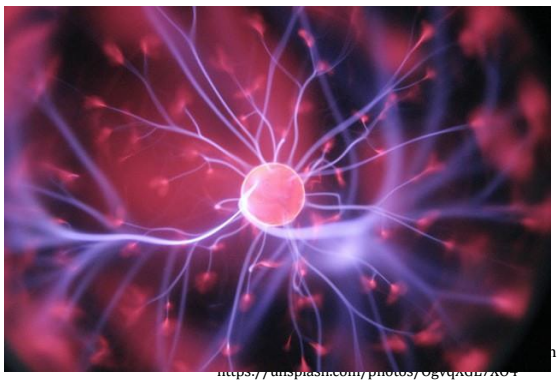
Problema A. Energía atómica

Nombre del archivo fuente: Atomic.c, Atomic.cpp, Atomic.java, Atomic.py Entrada: Estándar
Salida: Estándar

El *Next Wave Energy Research Club* está estudiando varios átomos como posibles fuentes de energía y te ha pedido que hagas algunos cálculos para ver cuáles son los más prometedores.



Aunque un átomo se compone de varias partes, a efectos de este método sólo es relevante el número de neutrones del átomo[1]. En este método, se dispara una carga láser contra el átomo, que libera energía en un proceso formalmente denominado *explodificación*. La forma exacta en que se desarrolla este proceso depende del número de neutrones k:

- Si el átomo contiene $k \leq n$ neutrones, se con- vertirá en un_k julios de energía.
- Si el átomo contiene $k > n$ neutrones, se descompondrá en dos átomos con i y j neutrones respectivamente, satisfaciendo $i, j \geq 1$ e $i + j = k$. Estos dos átomos se explodificarán a su vez.

Cuando se explota un átomo con k neutrones, la energía total que se libera depende de la secuencia exacta de descomposiciones que se producen en el proceso de explosión. La física moderna no es lo suficientemente potente como para predecir exactamente cómo se descompondrá un átomo; sin embargo, para que la explosificación sea una fuente de energía fiable, necesitamos conocer la cantidad mínima de energía que puede liberar al explotar. Se te ha encomendado la tarea de calcular esta cantidad.

Entrada

La entrada consiste en:

- Una línea con dos enteros n y q ($1 \leq n \leq 100$, $1 \leq q \leq 10^5$), el umbral de neutrones y el número de experimentos.
- Una línea con n enteros $a_1, \ldots, a_n$ ($1 \leq a_i \leq 10^9$ por cada i), donde a_i es la cantidad de energía liberada al explodificar un átomo con i neutrones.
- Luego siguen q líneas, cada una con un número entero k ($1 \leq k \leq 10^9$), preguntando por la mínima energía liberada al explodificar un átomo con k neutrones.

Salida

Para cada consulta k, obtenga la energía mínima liberada al explodificar un átomo con k neutrones.

[1] De hecho, para este problema quizá quieras olvidar todo lo que creías saber sobre química.

Ejemplo

Entrada	Salida
4 5 2 3 5 7 2 3 5 6 8	3 5 8 10 13
1 3 10 1 2 100	10 20 1000

Problema B. Bulldozer

Nombre del archivo fuente:Bulldozer .c, Bulldozer.cpp, Bulldozer.java, Bulldozer.py Entrada: Estándar
Salida: Estándar

La tarea consiste en derribar algunos edificios situados a lo largo de una carretera larga y recta. Los edificios están modelados como pilas uniformemente espaciadas de bloques cuadrados idénticos a lo largo de una línea infinita. Tu potente excavadora es capaz de mover cualquiera de estos bloques una unidad de distancia a la izquierda o a la derecha. Esto puede empujar a otros bloques fuera del camino, y los bloques que se encuentran encima de los bloques en movimiento se moverán a lo largo. Los bloques que son empujados por un hueco caen hacia abajo hasta llegar al suelo o a otro bloque.

Por ejemplo, considere las pilas de bloques que se muestran a la izquierda en la Figura 1, a continuación. Si empujas el bloque C hacia la derecha, los bloques D y E se desplazarán hacia la derecha, ya que están en medio. Los bloques A, B y F también se desplazarían porque están encima de los bloques en movimiento. Después de empujar C hacia la derecha, E quedaría encima de un hueco, por lo que E y F bajarían para rellenarlo. Las pilas resultantes se muestran en el centro de la Figura 1. Si empujamos el bloque C un paso más a la derecha, obtendremos la configuración que se muestra a la derecha.

Tu objetivo es *nivelar* todos los edificios: arrasa hasta que todas las pilas tengan una altura máxima de 1, es decir, que todos los bloques estén en el suelo. Ten en cuenta que la carretera se extiende infinitamente a ambos lados, así que esto siempre es posible.

Dadas las alturas iniciales de las pilas, determina el menor número de movimientos que necesitas hacer para nivelar todos los edificios, donde un movimiento consiste en usar el bulldozer para empujar un bloque un paso a la izquierda o a la derecha.

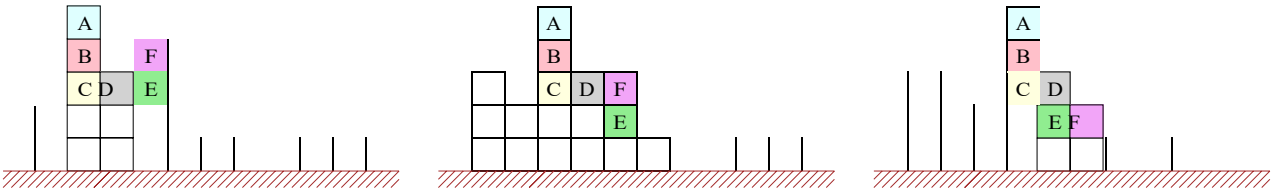


Figura 1: Ilustración de una configuración de pilas de bloques, y los resultados de empujar el bloque etiquetado como C hacia la derecha dos veces (los bloques etiquetados como A-F están coloreados y etiquetados sólo con fines ilustrativos).

Entrada

La entrada consiste en:

- Una línea con un número entero n ($1 \le n \le 2 \cdot 10^5$), el número de pilas de bloques.
- Una línea con n enteros $a_1, \ldots, a_n$ ($0 \le a_i \le 10^9$ para cada i), las alturas iniciales de las pilas de izquierda a derecha.

El ejemplo mostrado a la izquierda en la Figura 1 podría estar dado por 3, 5, 3, 4, 1, 1, 0, 0, 1, 1, pero también podría estar rellenado a la izquierda o a la derecha con ceros adicionales.

Salida

Indica el número mínimo de movimientos necesarios para nivelar cada edificio.

Ejemplo

Entrada	Salida
5 1 1 2 1 1	2
5 1 4 3 1 1	7
9 1 0 0 0 6 0 0 0 1	5
10 1 3 0 0 1 9 1 1 1 1	13

Problema C. Luchas en el concurso

Nombre del archivo fuente: Contest.c, Contest.cpp, Contest.java, Contest.py Entrada: Estándar
Salida: Estándar

Lotte participa en un concurso de programación. Su equipo ya ha resuelto k de los n problemas planteados, pero a medida que los problemas se vuelven más difíciles, Lotte empieza a perder la concentración y su mente empieza a divagar.

Globos de Pexels, Pixabay
https://pixabay.com/images/id-1869790/

Recuerda haber oído a los jueces hablar de la dificultad de los problemas, que califican en una escala entera de 0 a 100, ambos inclusive. De hecho, uno de los jueces dijo que "*el conjunto de problemas nunca había sido tan difícil, ¡la dificultad media de los problemas del conjunto de problemas es d!* "

Se pone a pensar en los problemas que su equipo ha resuelto hasta ahora y calcula su dificultad media. Con la esperanza de ganar algo de motivación, Lotte se pregunta si puede utilice esta información para determinar la dificultad media de los problemas restantes.

Entrada

La entrada consiste en:

- Una línea con dos números enteros n y k ($2 \le n \le 10^6$, $0 < k < n$), el número total de problemas y el número de problemas que el equipo de Lotte ha resuelto hasta ahora.
- Una línea con dos enteros d y s ($0 \le d, s \le 100$), la dificultad media de todos los problemas y la estimación de Lotte de la dificultad media de los problemas que ha resuelto su equipo.

Salida

Suponiendo que la estimación de Lotte es correcta, obtenga la dificultad media de los problemas sin resolver, o "imposible" si la dificultad media no existe. Su respuesta debe tener un error absoluto de como máximo 10^{-6}.

Ejemplo

Entrada	Salida
2 1 70 50	90.00
10 3 80 90	75.7142857
2 1 100 10	imposible

www.facebook.com/RedProg

ramacionCompetitivaTwitter: @RedProgramacion

Problema D. Círculo Dyson

Nombre del archivo fuente:Dyson .c, Dyson.cpp, Dyson.java, Dyson.py Entrada: Estándar
Salida: Estándar

Una Esfera de Dyson es una construcción teórica alrededor del Sol o de otra estrella, que captura toda la producción energética de la estrella. Los escritores de ciencia ficción han especulado con la posibilidad de que civilizaciones avanzadas acaben construyendo una esfera de este tipo, ya que las demandas energéticas de una sociedad así siguen creciendo sin límites. En nuestro espacio tridimensional, las esferas de Dyson siguen en el terreno de la ficción. *Dy & Son*, la principal empresa energética de sus vecinos dimensionales, le ha encargado la realización de un estudio de viabilidad en el mundo bidimensional de Flatland.

Dy & Son ha desarrollado un Dyson Circle modular. Consta de Unidades Dyson cuadradas independientes que pueden encadenarse para formar un bucle cerrado que acumula energía. Tu tarea consiste en averiguar cuántas de estas unidades Dyson necesitan para encerrar la estrella o estrellas que les interesan. Ten en cuenta que quieren un único Círculo Dyson, no uno separado para cada estrella.

Por comodidad, tanto las estrellas que Dy & Son quiere encerrar como las Unidades Dyson utilizadas para ello se modelan como cuadrados de exactamente 1 por 1 *Unidad Intergaláctica*, alineados con el *Sistema de Coordenadas Intergalácticas*. Sus unidades Dyson se conectan si tienen al menos una esquina en común. Véase un ejemplo en la Figura 2.

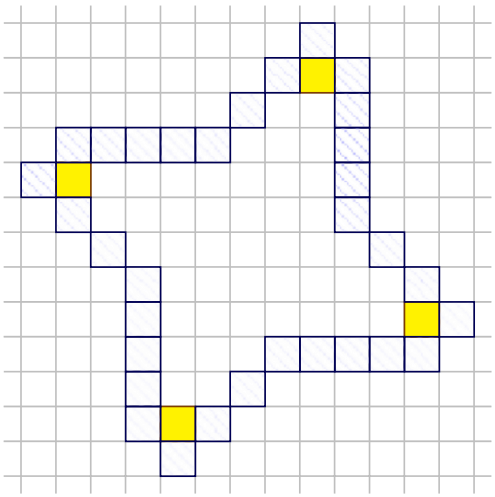


Figura 2: Ilustración del Ejemplo de Entrada 1: cuatro estrellas (cuadrados amarillos) y un Círculo de Dyson óptimo (cuadrados azules discontinuos) rodeándolas, y la negrura restante del espacio mostrada en blanco.

Formalmente, selecciona algunos cuadrados del plano para convertirlos en Unidades Dyson, de manera que los cuadrados restantes puedan dividirse en cuadrados *interiores* y *exteriores*. Todas las estrellas deben ser cuadrados interiores. Los cuadrados interiores deben formar una región contigua (conectada mediante aristas) y no conectarse con los cuadrados exteriores (mediante aristas). Los cuadrados exteriores forman una región contigua que se extiende hasta el infinito.

Entrada

La entrada consiste en:

- Una línea con un número entero n ($1 \le n \le 2 \cdot 10^5$), el número de estrellas.
- n líneas, cada una de las cuales contiene dos números enteros x e y ($-10^6 \le x, y \le 10^6$), la

ubicación del centro de una estrella.

No hay dos estrellas en el mismo lugar.

Salida

Produce el menor número de unidades Dyson necesarias para capturar la energía de todas las estrellas de la entrada.

Ejemplo

Entrada	Salida
4 2 5 -5 2 -2 -5 5 -2	32
2 1 1 3 2	8
2 2 3 4 5	9

www.facebook.com/RedProg

ramacionCompetitivaTwitter: @RedProgramacion

Problema E. Erupción

Nombre del archivo fuente:Eruption .c, Eruption.cpp, Eruption.java, Eruption.py Entrada: Estándar
Salida: Estándar

Un volcán ha entrado recientemente en erupción en Geldingadalur, Islandia. Afortunadamente, esta erupción es relativamente pequeña y, a diferencia de la famosa erupción del Eyjafjallajökull, no se espera que provoque una interrupción de los vuelos internacionales ni una conmoción mundial.

Erupción del Geldingadalur por Jon Schow, CC BY-SA

Existe cierta preocupación por el gas magmático que se ha liberado como parte de la erupción, ya que podría suponer un peligro para las poblaciones humanas de los alrededores. Los científicos han calculado la cantidad total de gas emitido, que, debido a la falta de viento, se ha extendido uniformemente por una zona circular alrededor de el centro del volcán. Las autoridades han evacuado la zona y ahora quieren cerrarla rodeando el perímetro con cinta de barrera.

Entrada

La entrada consiste en:

- Una línea con un número entero a ($1 \leq a \leq 10^{18}$), el área total cubierta por gas en metros cuadrados.

Salida

Calcula la longitud total de la cinta de barrera necesaria para rodear la zona cubierta por gas, en metros. Tu respuesta debe tener un error absoluto de 10^{-6} como máximo.

Ejemplo

Entrada	Salida
50	25.066282746
1234	124.526709336

www.redprogramacioncompetitiva.com/Twitter: @RedProgramacion

Problema F. Olimpiadas planas

Nombre del archivo fuente: Flatland.c,Flatland.cpp, Flatland.java, Flatland.py Input: Estándar
Salida: Estándar

Juegos Olímpicos de Pekín 2008 por PhotoBobil

https://en.wikipedia.org/wiki/File:Usain_Bolt_winning.jpg

Es el día después de Olimpia y usted, como organizador, está contento de que *todo haya* salido bien en estos tiempos difíciles. Bueno, no todo. . . .

Desde esta mañana, los correos electrónicos llenan tu bandeja de entrada con quejas por la falta de visibilidad en la carrera más importante: los 100 metros lisos. Piden que les devuelvas el dinero o amenazan con denunciarte en las redes sociales. Para empeorar las cosas, los espectadores no sólo se han quejado una vez, sino que te han enviado un correo electrónico por cada persona que se ha quejado. ¡bloquearon su vista en algún momento de la carrera! Incluso escribieron múltiples correos electrónicos cuando dos o más personas bloquearon su vista al mismo tiempo. Y no solo eso, algunos visitantes se quejaron al patrocinador principal *Dy & Son*, que a su vez les ha instado a mejorar la situación.

Como espera que en los próximos Juegos Olímpicos se permita la entrada a un mayor número de visitantes, supone que habrá aún más quejas si no aborda esta cuestión. Si la situación se agrava demasiado, puede incluso perder a su patrocinador Dy & Son. Por lo tanto, decide contar de antemano el número de quejas. Para ello, modela la pista de atletismo como un segmento recto y cuenta el número máximo de quejas que podría recibir en función de los asientos de los visitantes. Dependiendo del número de quejas que espere, determinará si necesita rehacer los asientos o simplemente reconfigurar su bloqueador de spam e intentar encontrar un nuevo patrocinador.

Entrada

La entrada consiste en:

- Una línea que contiene cuatro enteros x_s , y_s , x_e e y_e ($|x_s|$, $|y_s|$, $|x_e|$, $|y_e| \leq 10^9$), donde $s = (x_s , y_s)$ es el punto inicial de la pista de carrera y $e = (x_e , y_e)$ es el punto final de la pista de carrera. Tanto s como *e pertenecen a la pista*.
- Una línea que contiene un número entero n ($1 \leq n \leq 10^5$), el número de visitantes.
- n líneas, cada una de las cuales contiene dos números enteros x e y ($|x|, |y| \leq 10^9$), donde (x, y) es la ubicación del asiento de un visitante.

Se garantiza que la pista tiene una longitud positiva, es decir, $s \neq e$. Además, se puede suponer que todos los visitantes están sentados en lugares distintos y que ningún visitante está sentado en la pista.

Salida

Emita el número total de reclamaciones que recibiría para el aforo dado.

www.facebook.com/RedProg
ramacionCompetitivaTwitter: @RedProgramacion

Ejemplo

Entrada	Salida
0 0 100 0 4 50 20 50 30 50 50 120 0	3
0 0 100 0 5 50 20 50 30 50 -20 50 -30 100 30	2

Explicación

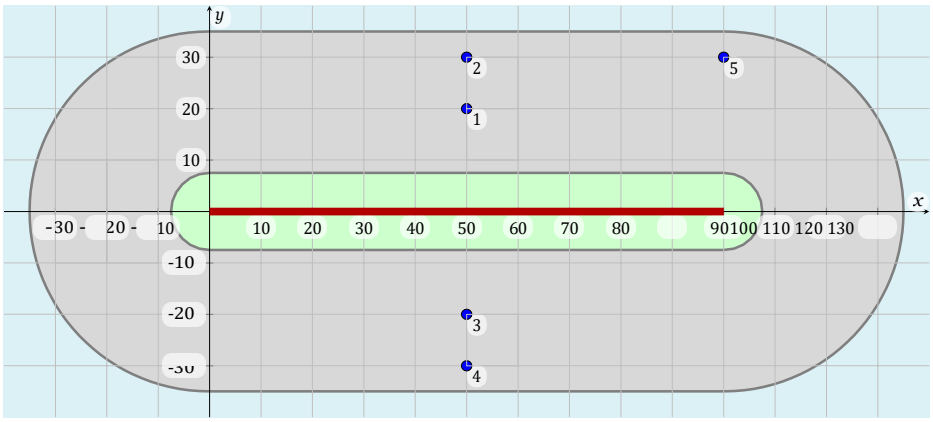


Figura 3: Ilustración del ejemplo de entrada 2. La pista de atletismo se dibuja como una línea roja y los asientos de los visitantes se resaltan en azul. El segundo visitante se quejará del primero y el cuarto se quejará del tercero.

etitiva.com/Twitter: @RedProgramacion

Problema G. Grandes esperanzas

Nombre del archivo fuente: Greatex.c,Greatex.cpp, Greatex.java, Greatex.py Entrada: Estándar
Salida: Estándar

Un *speedrun* es un recorrido por un juego con la intención de completarlo lo más rápido posible. Cuando se hace speedrunning, normalmente se sigue un camino previamente planificado a través del juego. A lo largo de este camino, puede haber algunos lugares en los que tengas que realizar una técnica difícil, o un *truco*, que puede causar un retraso si no lo consigues con éxito. Por suerte, puedes *reiniciar* el juego en cualquier momento: si has cometido algunos errores, puedes empezar un nuevo recorrido, perdiendo tu progreso pero empezando de cero instantáneamente. Puedes hacerlo tantas veces como quieras.

El juego que estás ejecutando actualmente tiene un récord de r segundos, que pretendes batir. Has descubierto un camino a través del juego que, en el mejor de los casos, lleva $n < r$ *segundos*. Sin embargo, hay algunos trucos en el camino: sabes exactamente en qué punto del recorrido se producen, cuál es la probabilidad de que los realices con éxito y cuántos segundos tienes que emplear para recuperarte si fallan.

Dados estos datos, quieres encontrar la estrategia óptima para cuando reiniciar el juego para minimizar el tiempo esperado para establecer un nuevo récord. Escribe un programa para determinar cuál es este menor tiempo esperado posible.

Entrada

La entrada consiste en:

- Una línea con tres números enteros n, r y m ($2 \le n < r \le 5\,000$, $1 \le m \le 50$), donde n y r son los descritos anteriormente y m es el número de bazas.
- m líneas, cada una de las cuales contiene tres números que describen un truco:
  - Un número entero t ($1 \le t < n$), el momento de la ruta (suponiendo que no haya fallado ninguna baza antes) en el que se produce la baza,
  - un número real p ($0 < p < 1$ y p tiene como máximo 6 dígitos después del punto decimal), la probabilidad de que el truco tenga éxito, y
  - un número entero d ($1 \le d \le 1\,000$), el número de segundos necesarios para recuperarse en caso de que falle el truco.

Las bazas se dan ordenadas por t, y no hay dos bazas que ocurran al mismo tiempo t en la ruta.

Puede suponer que, sin reiniciar, una sola partida tiene una probabilidad de al menos 1 entre $50\,000$ de conseguir mejorar el récord.

Salida

Obtenga el tiempo esperado que tendrá que jugar el juego para establecer un nuevo récord, suponiendo que se utiliza una estrategia óptima. Tu respuesta debe tener un error absoluto de como máximo 10^{-6} .

www.facebook.com/RedProg
ramacionCompetitivaTwitter: @RedProgramacion

Ejemplo

Entrada	Salida
100 111 5 20 0.5 10 80 0.5 2 85 0.5 2 90 0.5 2 95 0.5 2	124
2 4 1 1 0.5 5	3
10 20 3 5 0.3 8 6 0.8 3 8 0.9 3	18.9029850746
10 50 1 5 0.5 30	15

Explicación

Ejemplo Entrada 1:

El récord de este juego es de 111 segundos, y tu ruta dura 100 segundos si todo va bien.

Después de jugar 20 segundos, hay un truco con un porcentaje de éxito del 50%. Si tiene éxito, sigues jugando. Si falla, pierdes 10 segundos: ahora la carrera durará al menos 110 segundos. Sigue siendo posible batir un récord, pero todas las demás bazas de la carrera tienen que tener éxito. Resulta que, por término medio, es más rápido reiniciar después de fallar la primera baza.

Así, repite los primeros 20 segundos del juego hasta que el truco tiene éxito: con probabilidad 1/2, tarda 1 intento; con probabilidad 1/4, tarda 2 intentos; y así sucesivamente. Por término medio, empleas 40 segundos en los primeros 20 segundos del recorrido.

Una vez que has realizado con éxito el primer truco, quieres terminar la carrera sin importar el resultado de los otros trucos: se tarda 80 segundos, más una media de 1 segundo de pérdida de cada uno de los 4 trucos restantes. Por lo tanto, el tiempo previsto hasta batir el récord es de 124 segundos.

etitiva.com/Twitter: @RedProgramacion

Problema H. Calentamiento

Nombre del archivo fuente: Calefaccion.c, Calefaccion.cpp, Calefaccion.java, Calefaccion.py Entrada: Estándar
Salida: Estándar

Jonas acaba de participar en su primer concurso de comer guindillas. Se le presenta una pizza compuesta por n porciones, numeradas del 1 al n, cada una de las cuales contiene una selección de guindillas. Inicialmente, las porciones i e $i + 1$ son adyacentes en el plato (donde $1 \leq i < n$), y también lo son las porciones 1 y n. Según las reglas del concurso, sólo se puede consumir una porción a la vez, y la porción debe terminarse en su totalidad antes de empezar una nueva porción. Jonas puede elegir cualquier rebanada para comer primero, pero después sólo puede comer rebanadas que tengan como máximo una rebanada adyacente restante.

Chilli pizza por Rahul Upadhyay, Unsplash
https://unsplash.com/photos/yDKHJxfiWDk

El picante de cada loncha se mide en unidades Scoville (SHU). Jonas tiene una cierta tolerancia al picante, también medida en SHU, que corresponde al picante de la loncha más picante que Jonas puede tolerar comer. También ha observado que, tras comer una loncha de k *SHU*, su tolerancia aumenta inmediatamente en k.

Para ganar el concurso, a Jonas le gustaría terminarse todas las porciones de su pizza. Ayúdale a determinar la tolerancia inicial mínima de picante necesaria para hacerlo respetando las reglas del concurso.

Entrada

La entrada consiste en:

- Una línea con un número entero n ($3 \leq n \leq 5 \cdot 10^5$), el número de porciones de pizza.
- Una línea con n enteros $s_1, s_2, \ldots, s_n$ ($0 \leq s_i \leq 10^{13}$), donde s_i es el picante de la i-ésima loncha en SHU.

Salida

Indica la tolerancia inicial mínima al picante en SHU que necesita Jonas para poder comerse todos los trozos de pizza.

Ejemplo

Entrada	Salida
5 5 0 10 6 1	4
7 20 23 7 2 3 7 1	2

www.facebook.com/RedProg
ramacionCompetitivaTwitter: @RedProgramacion

Problema I. Corporación Islandesa de Circulación de Parcelas

Nombre del archivo fuente: islandic.c, Icelandic.cpp, Icelandic.java, Icelandic.py Entrada: Estándar
Salida: Estándar

La *Icelandic Corporation for Parcel Circulation* es la principal empresa de transporte de mercancías entre Islandia y el resto del mundo. Su última innovación es un enlace de drones que conecta con Europa continental y que cuenta con varios drones que viajan de un lado a otro por una misma ruta.

Drone por Hyeri Kim, Pixabay
https://pixabay.com/images/id-1134764/

Los drones están equipados con un sofisticado sistema que les permite realizar maniobras evasivas cuando dos drones se acercan. Desgraciadamente, un fallo de software ha hecho que este sistema se averíe y ahora todos los drones vuelan a lo largo de la ruta sin forma de evitar colisiones entre ellos.

A efectos de este problema, los drones se consideran puntos que se mueven a lo largo de una línea recta infinita con velocidad constante. Siempre que dos drones se encuentren en el mismo punto, se colisionarán, lo que provocará que se salgan de su trayectoria de vuelo y caigan en picado en el Océano Atlántico. Se garantiza que el programa de vuelo de los drones sea tal que en ningún momento haya tres o más drones colisionando en el mismo lugar.

Conoces la posición actual de cada dron, así como sus velocidades. Tu tarea consiste en evaluar los daños causados por el fallo del sistema averiguando qué drones seguirán volando indefinidamente sin estrellarse.

Entrada

La entrada consiste en:

- Una línea con un número entero n ($1 \le n \le 10^5$), el número de zánganos. Los zánganos se numeran a partir de 1 a n.
- n *rectas*, la *i-ésima* de las cuales contiene dos enteros x_i y v_i ($-10^9 \le x_i, v_i \le 10^9$), la posición actual y la velocidad del *i-ésimo* zángano a lo largo de la recta infinita.

Los zánganos están dados por coordenadas x crecientes y no hay dos zánganos en la misma posición, es decir $x_i < x_{i+1}$ para cada i. Puede suponer que nunca habrá una colisión en la que participen tres o más zánganos.

Salida

La salida consiste en:

- Una línea con un entero m ($0 \le m \le n$), el número de drones que nunca se estrellan.
- Una línea con m enteros separados por espacios, los índices de los drones que nunca se estrellan en orden numérico creciente.

Ejemplo

Entrada	Salida
3 10 15 30 5 50 -1	1 3
6 0 3 2 2 3 1 4 3 5 2 6 3	2 1 6
10 -8 1 -4 1 -3 3 -2 -9 2 -3 4 1 5 1 6 4 8 4 10 3	4 1 6 7 8

ramacionCompetitivaTwitter: @RedProgramacion

Problema J. Jet Set

Nombre del archivo fuente: Jetset.c, Jetset.cpp, Jetset.java, Jetset.py Entrada: Estándar
Salida: Estándar

A tus amigos ricos les encanta presumir de sus viajes. Cada vez que los ves, han visitado algún lugar exótico del que nunca habías oído hablar. Todos ellos están encantados de decirte que han dado *la* vuelta al mundo, pero tú no estás tan seguro. ¿Han *dado* realmente la vuelta al mundo?

Existen muchas definiciones diferentes de lo que constituye exactamente una circunnavegación, pero a efectos de este problema consideramos que una circunnavegación es un viaje que comienza y termina en el mismo punto y que visita todos los meridianos (líneas de longitud) a lo largo del camino. El Polo Norte y el Polo Sur forman parte de cada meridiano.

Ilustración del Ejemplo de Entrada 1, con una circunnavegación que comienza y termina en Reikiavik, con waypoints adicionales en Atenas, Yakarta, Honolulú y Chicago.

Amelia, una de tus amigas ricas, te dio un registro de sus vuelos en forma de lista de waypoints. Su viaje empezaba en el primer waypoint, visitaba los restantes en orden y, por último, volvía del último waypoint al primero. Entre waypoints consecutivos, Amelia siempre viajaba a lo largo del arco circular más corto que conectaba los dos puntos. Averigua si el viaje de Amelia puede considerarse una circunnavegación en el sentido anterior y, en caso negativo, encuentra un meridiano que Amelia nunca visitó.

Entrada

La entrada consiste en:

- Una línea con un número entero n ($2 \leq n \leq 1\,000$), el número de waypoints.
- n líneas, cada una con dos enteros φ y λ ($-90 < \varphi < 90$, $-180 \leq \lambda < 180$), la latitud y la longitud de uno de los waypoints.

No hay dos waypoints consecutivos a lo largo de la ruta que sean iguales o antípodas (puntos opuestos en la esfera) entre sí.

Salida

Si la ruta es una circunnavegación válida, la salida es sí. En caso contrario, no.

Ejemplo

Entrada	Salida
5 64 -22 38 24 -6 107 21 -158 42 -88	sí
2 80 30 75 -150	sí
4 45 0 0 -170 -45 0 0 170	no

ramacionCompetitivaTwitter: @RedProgramacion

Problema K. Knitpicking

Nombre del archivo fuente: Knitpicking .c, Knitpicking.cpp, Knitpicking.java, Knitpicking.py Entrada: Estándar
Salida: Estándar

Kattis tiene en su cajón muchos pares de calcetines de punto bonitos y calentitos, perfectos para el invierno. Estos calcetines vienen en una amplia gama de colores y tipos, y se han mezclado todos juntos. Cada mañana, Kattis tiene que elegir dos calcetines iguales.

Para encontrar calcetines iguales, simplemente saca calcetines sueltos al azar del cajón hasta que encuentra un par igual. Puede llevarle mucho tiempo, por ejemplo, si sigue sacando calcetines de la derecha sin encontrar el par de la izquierda. ¿Cuánto tiempo necesita
dibujando calcetines hasta que se le garantice que tiene un par que ponerse?

Gillie en uno de sus lugares de descanso Por Dwight Sipler (cc by-sa)

Entrada

La entrada consiste en:

- Una línea con un número entero n ($1 \leq n \leq 1\,000$), el número de grupos de calcetines idénticos.
- n líneas, cada una de las cuales describe un grupo de calcetines idénticos con lo siguiente:
  - Una cadena i, el tipo de los calcetines del grupo. El tipo i consta de entre 1 y 20 letras inglesas minúsculas. Los calcetines del mismo tipo se consideran compatibles a efectos de moda.
  - Una cadena j, el ajuste de los calcetines del grupo, que puede ser izquierdo, derecho o cualquiera, indicando si los calcetines se ajustan al pie izquierdo, al pie derecho o a cualquier pie.
  - Un número entero k ($1 \leq k \leq 1\,000$), el número de calcetines del cajón que son de este tipo y talla.

Un ajuste dado de un tipo dado de calcetín aparece como máximo una vez en la entrada.

Salida

Indica el número mínimo de calcetines que Kattis debe sacar para obtener un par. Si no es posible obtener un par, el resultado es imposible.

Ejemplo

Entrada	Salida
3 difusa cualquier 10 lana izquierda 6 lana derecha 4	8
3 deportes cualquier 1 negro izquierda 6 blanco derecha 6	imposible

2 calentar cualquier 5 izquierda caliente 3	4

Problema L. Camisa de la suerte

Nombre del archivo fuente: Lucky.c, Lucky.cpp, Lucky.java, Lucky.py Input: Estándar
Salida: Estándar


Pila de ropa via pxfuel.com

Usted, un frenético programador competitivo, ha coleccionado un gran número de camisetas compitiendo en diversas pruebas de programación. De hecho, tienes tantas que son las únicas camisetas que te pones. Las guardas cuidadosamente dobladas en una gran pila en tu enorme armario. Cada mañana, eliges la camiseta que más te gusta de la pila para ponértela ese día. Al final del día, la tiras al cesto de la ropa sucia.

Para tener siempre camisas limpias, a veces también se hace la colada por la noche, lavando todas las camisas del cesto de la ropa sucia (incluida la que se ha puesto ese día). Sin embargo, esto no se hace de acuerdo con un horario ordenado; el número de días entre tus ciclos de lavado es un número entero uniformemente aleatorio entre 1 (en cuyo caso lavarías una sola camisa) y el número de camisas que tengas. Después de lavar la ropa, la vuelves a poner encima de la pila en un orden uniformemente aleatorio.

Es la noche después de un exitoso concurso de programación, y decides que la camiseta que conseguiste allí es tu camiseta de la suerte a partir de ahora. Te preguntas cuándo podrás volver a ponértela, y te entretienes pensando en la buena suerte que recibirás cuando lo hagas. Acabas de terminar de hacer la colada y pones todas las camisetas en la pila. Conociendo la posición actual de tu camisa de la suerte en la pila, ¿cuál es la posición esperada de tu camisa de la suerte después de k ciclos de lavado más?

Entrada

La entrada consiste en:

- Una línea que contiene tres números enteros n ($1 \leq n \leq 10^6$), el número de camisas que tienes, i ($1 \leq i \leq n$), la posición de tu camisa de la suerte contada desde arriba, y k ($1 \leq k \leq 10^6$), el número de ciclos de lavado.

Salida

Obtenga la posición esperada de su camisa de la suerte después de k ciclos de lavado. Su respuesta debe tener un error absoluto de como máximo 10^{-6}.

Ejemplo

Entrada	Salida
3 2 2	1.833333333
5 1 1	2.0
10 7 100	5.499986719

www.facebook.com/RedProgramacionCompetitivaTwitter: @RedProgramacion